Feather ESP32-S3 TFT CircuitPython Day 2024 Countdown Clock

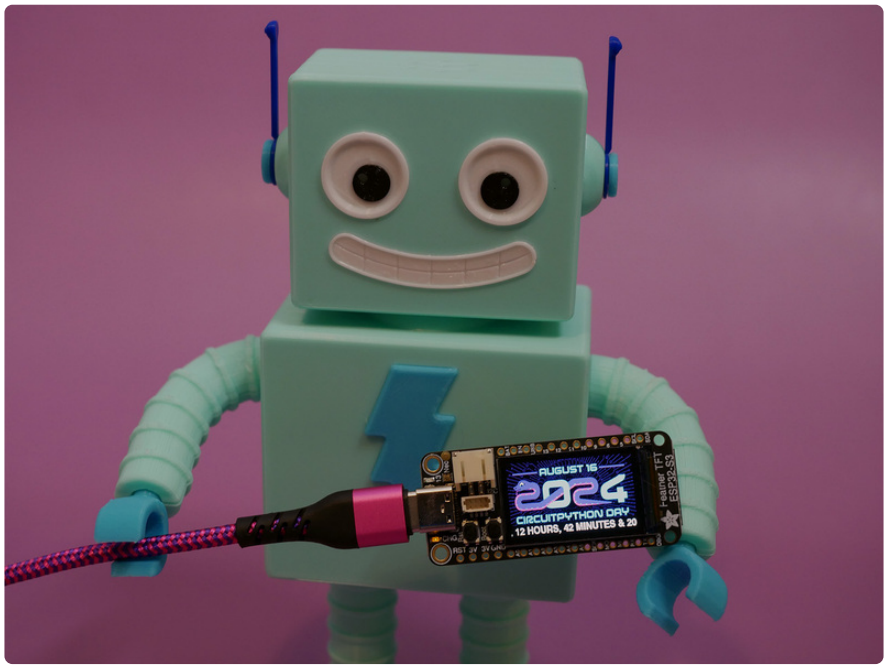
Created by Liz Clark

https://learn.adafruit.com/feather-esp32-s3-tft-circuitpython-day-2024-countdown-clock

Last updated on 2024-11-18 01:00:41 PM EST

Table of Contents

Overview

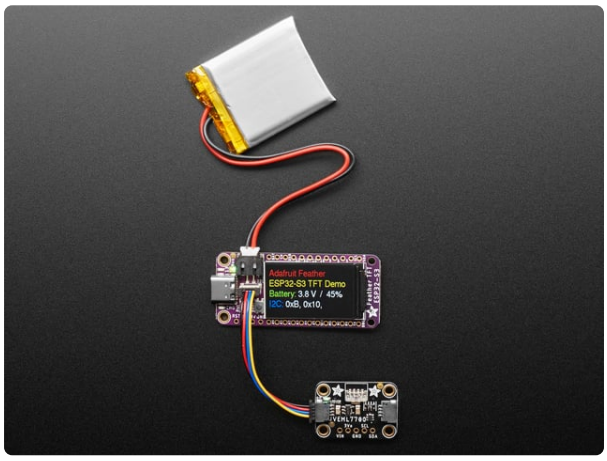


In this year of chaos and calamity, one day on the horizon fills us with hope: CircuitPython Day 2024! Countdown to the snakiest day of the year (August 16, 2024) with, what else? A CircuitPython project!

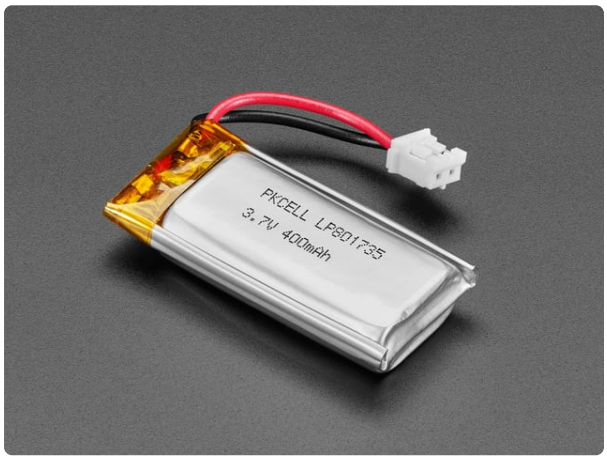
In this project, you'll use a Feather ESP32-S3 TFT board to get time from the internet and calculate down to the second how much longer you have to wait for this year's festivities.

You can power the project with a lipo battery or USB - maker's choice.

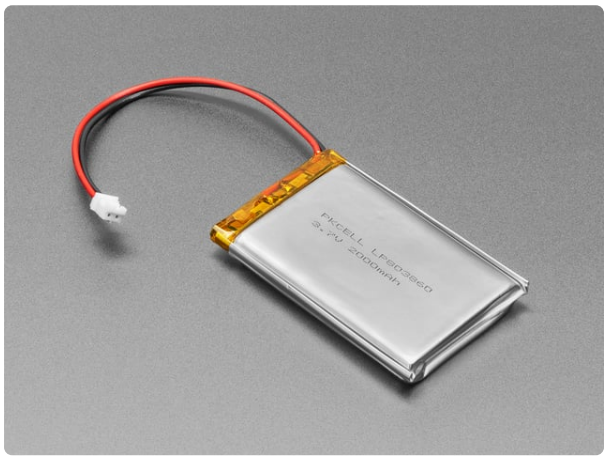
Parts



[Adafruit ESP32-S3 TFT Feather - 4MB Flash, 2MB PSRAM, STEMMA QT]
We've got a new machine here at Adafruit, it can uncover your deepest desires. Don't believe me? I'll turn it on right now to prove it to you! What, you want your very own...
https://www.adafruit.com/product/5483



[Lithium Ion Polymer Battery Ideal For Feathers - 3.7V 400mAh]
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/3898



[Lithium Ion Battery - 3.7V 2000mAh]
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This...
https://www.adafruit.com/product/2011

Pink and Purple Woven USB A to USB C Cable - 1 meter long

This cable is not only super-fashionable, with a woven pink and purple Blinka-like pattern, it's also made for USB C for our modernized breakout boards, Feathers, and...

https://www.adafruit.com/product/5153

CircuitPython

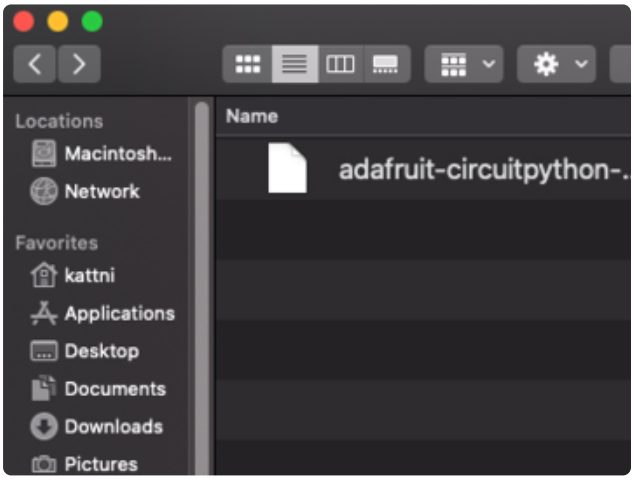
CircuitPython (https://adafru.it/tB7) is a derivative of MicroPython (https://adafru.it/BeZ) designed to simplify experimentation and education on low-cost microcontrollers. It makes it easier than ever to get prototyping by requiring no upfront desktop software downloads. Simply copy and edit files on the **CIRCUITPY** drive to iterate.

CircuitPython Quickstart

Follow this step-by-step to quickly get CircuitPython running on your board.

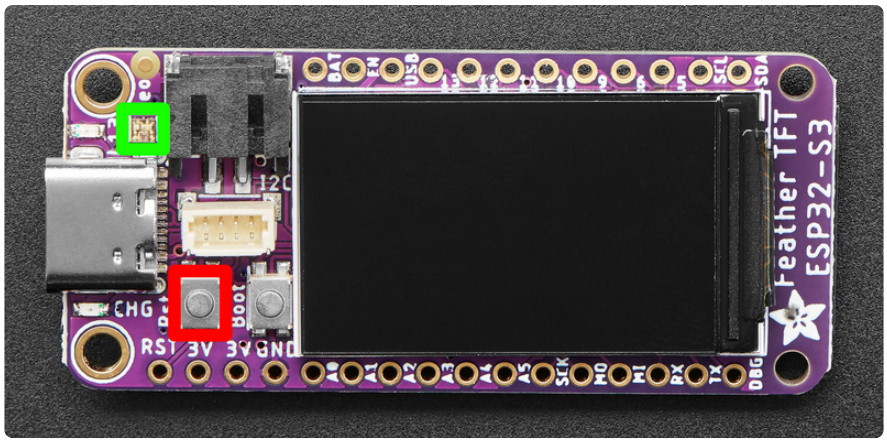
Download the latest version of CircuitPython for this board via circuitpython.org

https://adafru.it/10yc



Click the link above to download the latest CircuitPython UF2 file.

Save it wherever is convenient for you.



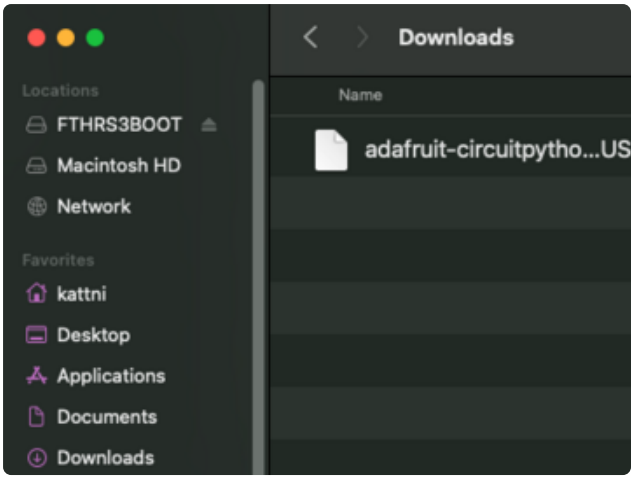
Plug your board into your computer, using a known-good data-sync cable, directly, or via an adapter if needed.

Double-click the **reset** button (highlighted in red above), and you will see the **RGB status LED(s)** turn green (highlighted in green above). If you see red, try another port, or if you're using an adapter or hub, try without the hub, or different adapter or hub.

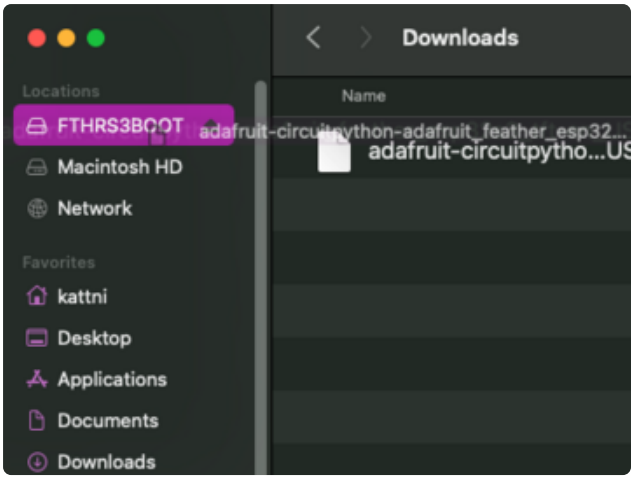
For this board, **tap reset and wait for the LED to turn purple, and as soon as it turns purple, tap reset again**. The second tap needs to happen while the LED is still purple.

If double-clicking doesn't work the first time, try again. Sometimes it can take a few tries to get the rhythm right!

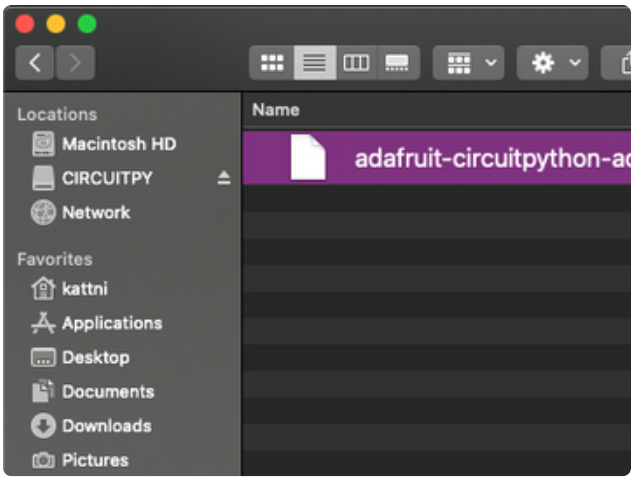
A lot of people end up using charge-only USB cables and it is very frustrating! **Make sure you have a USB cable you know is good for data sync.**



You will see a new disk drive appear called **FTHRS3BOOT**.



Drag the **adafruit_circuitpython_etc.uf2** file to **FTHRS3BOOT**.



The **BOOT** drive will disappear and a new disk drive called **CIRCUITPY** will appear.

That's it!

Create Your settings.toml File

CircuitPython works with WiFi-capable boards to enable you to make projects that have network connectivity. This means working with various passwords and API keys. As of CircuitPython 8 (https://adafru.it/Em8), there is support for a **settings.toml** file. This is a file that is stored on your **CIRCUITPY** drive, that contains all of your secret network information, such as your SSID, SSID password and any API keys for IoT

services. It is designed to separate your sensitive information from your **code.py** file so you are able to share your code without sharing your credentials.

CircuitPython previously used a **secrets.py** file for this purpose. The **settings.toml** file is quite similar.

> Your settings.toml file should be stored in the main directory of your CIRCUITPY drive. It should not be in a folder.

CircuitPython **settings.toml** File

This section will provide a couple of examples of what your **settings.toml** file should look like, specifically for CircuitPython WiFi projects in general.

The most minimal **settings.toml** file must contain your WiFi SSID and password, as that is the minimum required to connect to WiFi. Copy this example, paste it into your **settings.toml**, and update:

- `your_wifi_ssid`
- `your_wifi_password`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
```

Many CircuitPython network-connected projects on the Adafruit Learn System involve using Adafruit IO. For these projects, you must also include your Adafruit IO username and key. Copy the following example, paste it into your settings.toml file, and update:

- `your_wifi_ssid`
- `your_wifi_password`
- `your_aio_username`
- `your_aio_key`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_AIO_USERNAME = "your_aio_username"
ADAFRUIT_AIO_KEY = "your_aio_key"
```

Some projects use different variable names for the entries in the **settings.toml** file. For example, a project might use `ADAFRUIT_AIO_ID` in the place of

`ADAFRUIT_AIO_USERNAME`. **If you run into connectivity issues, one of the first things to check is that the names in the settings.toml file match the names in the code.**

> Not every project uses the same variable name for each entry in the settings.toml file! Always verify it matches the code.

settings.toml File Tips

Here is an example **settings.toml** file.

```
# Comments are supported
CIRCUITPY_WIFI_SSID = "guest wifi"
CIRCUITPY_WIFI_PASSWORD = "guessable"
CIRCUITPY_WEB_API_PORT = 80
CIRCUITPY_WEB_API_PASSWORD = "passw0rd"
test_variable = "this is a test"
thumbs_up = "\U0001f44d"
```

In a **settings.toml** file, it's important to keep these factors in mind:

- Strings are wrapped in double quotes; ex: `"your-string-here"`
- Integers are **not** quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
    - Floats, octal (`0o567`) and binary (`0b11011`) are not supported.
- Use `\u` escapes for weird characters, `\x` and `\ooo` escapes are not available in **.toml** files
    - Example: `\U0001f44d` for (thumbs up emoji) and `\u20ac` for € (EUR sign)
- Unicode emoji, and non-ASCII characters, stand for themselves as long as you're careful to save in "UTF-8 without BOM" format

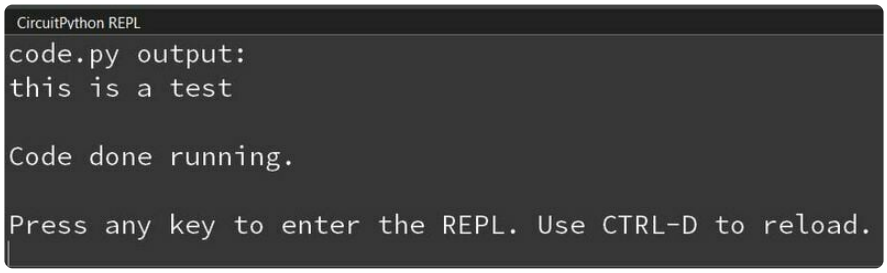
When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

Accessing Your **settings.toml** Information in **code.py**

In your **code.py** file, you'll need to `import` the `os` library to access the **settings.toml** file. Your settings are accessed with the `os.getenv()` function. You'll pass your settings entry to the function to import it into the **code.py** file.

```
import os

print(os.getenv("test_variable"))
```



In the upcoming CircuitPython WiFi examples, you'll see how the **settings.toml** file is used for connecting to your SSID and accessing your API keys.

Code the Countdown Clock

Once you've finished setting up your Feather ESP32-S3 TFT with CircuitPython, you can access the code and necessary libraries by downloading the Project Bundle.

To do this, click on the **Download Project Bundle** button in the window below. It will download to your computer as a zipped folder.

```
# SPDX-FileCopyrightText: 2024 Liz Clark for Adafruit Industries
#
# SPDX-License-Identifier: MIT

import os
import time
import wifi
```

```
import board
import displayio
import socketpool
import microcontroller
from adafruit_bitmap_font import bitmap_font
from adafruit_display_text import bitmap_label
import adafruit_ntp
from adafruit_ticks import ticks_ms, ticks_add, ticks_diff

timezone = -4
# The time of the thing!
EVENT_YEAR = 2024
EVENT_MONTH = 8
EVENT_DAY = 16
EVENT_HOUR = 0
EVENT_MINUTE = 0
# we'll make a python-friendly structure
event_time = time.struct_time((EVENT_YEAR, EVENT_MONTH, EVENT_DAY,
                               EVENT_HOUR, EVENT_MINUTE, 0,  # we don't track
seconds
                               -1, -1, False))  # we dont know day of week/year or
DST

wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
pool = socketpool.SocketPool(wifi.radio)
ntp = adafruit_ntp.NTP(pool, tz_offset=timezone, cache_seconds=3600)

display = board.DISPLAY
group = displayio.Group()
font = bitmap_font.load_font("/Helvetica-Bold-16.pcf")
blinka_bitmap = displayio.OnDiskBitmap("/cpday_tft.bmp")
blinka_grid = displayio.TileGrid(blinka_bitmap,
pixel_shader=blinka_bitmap.pixel_shader)
scrolling_label = bitmap_label.Label(font, text=" ", y=display.height - 13)

group.append(blinka_grid)
group.append(scrolling_label)
display.root_group = group
display.auto_refresh = False

refresh_clock = ticks_ms()
refresh_timer = 3600 * 1000
clock_clock = ticks_ms()
clock_timer = 1000
scroll_clock = ticks_ms()
scroll_timer = 50
first_run = True

while True:
    # only query the online time once per hour (and on first run)
    if ticks_diff(ticks_ms(), refresh_clock) >= refresh_timer or first_run:
        try:
            print("Getting time from internet!")
            now = ntp.datetime
            print(now)
            total_seconds = time.mktime(now)
            first_run = False
            refresh_clock = ticks_add(refresh_clock, refresh_timer)
        except Exception as e: # pylint: disable=broad-except
            print("Some error occured, retrying! -", e)
            time.sleep(2)
            microcontroller.reset()

    if ticks_diff(ticks_ms(), clock_clock) >= clock_timer:
        remaining = time.mktime(event_time) - total_seconds
        secs_remaining = remaining % 60
        remaining //= 60
        mins_remaining = remaining % 60
```

```
        remaining //= 60
        hours_remaining = remaining % 24
        remaining //= 24
        days_remaining = remaining
        scrolling_label.text = (f"{days_remaining} DAYS, {hours_remaining} HOURS," +
                                f"{mins_remaining} MINUTES & {secs_remaining}
SECONDS")
        total_seconds += 1
        clock_clock = ticks_add(clock_clock, clock_timer)
    if ticks_diff(ticks_ms(), scroll_clock) >= scroll_timer:
        scrolling_label.x -= 1
        if scrolling_label.x < -(scrolling_label.width + 5):
            scrolling_label.x = display.width + 2
        display.refresh()
        scroll_clock = ticks_add(scroll_clock, scroll_timer)
```

Upload the Code and Libraries to the Feather ESP32-S3 TFT

After downloading the Project Bundle, plug your Feather ESP32-S3 TFT into the computer's USB port with a known good USB data+power cable. You should see a new flash drive appear in the computer's File Explorer or Finder (depending on your operating system) called **CIRCUITPY**. Unzip the folder and copy the following items to the Feather ESP32-S3 TFT's **CIRCUITPY** drive.

- **lib** folder
- **code.py**
- **cpday_tft.bmp**
- **Helvetica-Bold-16.pcf**

Your Feather ESP32-S3 TFT **CIRCUITPY** drive should look like this after copying the **lib** folder, **cpday_tft.bmp** file, **Helvetica-Bold-16.pcf** file and the **code.py** file.

Add Your **settings.toml** File

As of CircuitPython 8.0.0, there is support for Environment Variables (https://adafru.it/11wE). Environment variables are stored in a **settings.toml** file. Similar to **secrets.py**, the **settings.toml** file separates your sensitive information from your main **code.py** file. Add your **settings.toml** file as described in the Create Your settings.toml File page (https://adafru.it/19ap) earlier in this guide. You'll need to include your `CIRCUITPY_WIFI_SSID` and `CIRCUITPY_WIFI_PASSWORD`.

```
CIRCUITPY_WIFI_SSID = "your-ssid-here"
CIRCUITPY_WIFI_PASSWORD = "your-ssid-password-here"
```

How the CircuitPython Code Works

At the top of the code, you'll edit `timezone` to reflect your UTC timezone offset for your location. The event time is also setup. In this case, it's August 16, 2024 at midnight.

```
timezone = -4

# The time of the thing!
EVENT_YEAR = 2024
EVENT_MONTH = 8
EVENT_DAY = 16
EVENT_HOUR = 0
EVENT_MINUTE = 0
# we'll make a python-friendly structure
event_time = time.struct_time((EVENT_YEAR, EVENT_MONTH, EVENT_DAY,
                               EVENT_HOUR, EVENT_MINUTE, 0,  # we don't track
seconds
                               -1, -1, False))  # we dont know day of week/year or
DST
```

WiFi and NTP

WiFi is setup along with an NTP instance. NTP (https://adafru.it/lgf) is a networking protocol for clock synchronization and will take care of the timing for this project. Your `timezone` is passed to the NTP instance to reflect the time in your location.

```
wifi.radio.connect(os.getenv("CIRCUITPY_WIFI_SSID"),
os.getenv("CIRCUITPY_WIFI_PASSWORD"))
pool = socketpool.SocketPool(wifi.radio)

ntp = adafruit_ntp.NTP(pool, tz_offset=timezone, cache_seconds=3600)
```

Graphics

Next are the display objects for the built-in TFT display on the Feather. This takes care of the background bitmap graphic and text element.

```
display = board.DISPLAY
group = displayio.Group()
```

```
font = bitmap_font.load_font("/Helvetica-Bold-16.pcf")
blinka_bitmap = displayio.OnDiskBitmap("/cpday_tft.bmp")
blinka_grid = displayio.TileGrid(blinka_bitmap,
pixel_shader=blinka_bitmap.pixel_shader)
scrolling_label = bitmap_label.Label(font, text=" ", y=display.height - 13)

group.append(blinka_grid)
group.append(scrolling_label)
display.root_group = group
display.auto_refresh = False
```

Time is Ticking

Finally, three separate `ticks` timers are created for timekeeping in the loop.

```
refresh_clock = ticks_ms()
refresh_timer = 3600 * 1000
clock_clock = ticks_ms()
clock_timer = 1000
scroll_clock = ticks_ms()
scroll_timer = 50
```

The Loop

In the loop, the time is fetched from the NTP server every hour and stored in `now`. `now` is converted to seconds using `time.mktime(now)`. This lets you calculate how much time is remaining until the event.

```
if ticks_diff(ticks_ms(), refresh_clock) &gt;= refresh_timer or first_run:
    try:
          print("Getting time from internet!")
          now = ntp.datetime
          print(now)
          total_seconds = time.mktime(now)
          first_run = False
          refresh_clock = ticks_add(refresh_clock, refresh_timer)
    except RuntimeError as e:
          print("Some error occured, retrying! -", e)
          continue
```

The time is kept by the microcontroller in between polling the NTP server. Every second, 1 second is added to the `total_seconds` value tracking the current time. `remaining` stores the total seconds remaining until the event. This is converted to days, hours, minutes and seconds. These values are added to the scrolling text on the TFT.

```
if ticks_diff(ticks_ms(), clock_clock) &gt;= clock_timer:
        remaining = time.mktime(event_time) - total_seconds
        secs_remaining = remaining % 60
        remaining //= 60
        mins_remaining = remaining % 60
        remaining //= 60
        hours_remaining = remaining % 24
        remaining //= 24
        days_remaining = remaining
        my_scrolling_label.text = f"{days_remaining} Days, {hours_remaining} Hours,
{mins_remaining} Minutes and {secs_remaining} Seconds"
```

```
        total_seconds += 1
        clock_clock = ticks_add(clock_clock, clock_timer)
```

The last timer is used to scroll the text by moving the `x` coordinate of the text by 2 pixels. When the text is offscreen, its `x` coordinate is reset to start scrolling across again.

```
if ticks_diff(ticks_ms(), scroll_clock) &gt;= scroll_timer:
        my_scrolling_label.x -= 2
        if my_scrolling_label.x &lt; -(my_scrolling_label.width + 5):
            my_scrolling_label.x = display.width + 2
        display.refresh()
        scroll_clock = ticks_add(scroll_clock, scroll_timer)
```

Create Your settings.toml File

CircuitPython works with WiFi-capable boards to enable you to make projects that have network connectivity. This means working with various passwords and API keys. As of CircuitPython 8 (https://adafru.it/Em8), there is support for a **settings.toml** file. This is a file that is stored on your **CIRCUITPY** drive, that contains all of your secret network information, such as your SSID, SSID password and any API keys for IoT services. It is designed to separate your sensitive information from your **code.py** file so you are able to share your code without sharing your credentials.

CircuitPython previously used a **secrets.py** file for this purpose. The **settings.toml** file is quite similar.

> Your settings.toml file should be stored in the main directory of your CIRCUITPY drive. It should not be in a folder.

CircuitPython **settings.toml** File

This section will provide a couple of examples of what your **settings.toml** file should look like, specifically for CircuitPython WiFi projects in general.

The most minimal **settings.toml** file must contain your WiFi SSID and password, as that is the minimum required to connect to WiFi. Copy this example, paste it into your **settings.toml**, and update:

- `your_wifi_ssid`
- `your_wifi_password`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
```

Many CircuitPython network-connected projects on the Adafruit Learn System involve using Adafruit IO. For these projects, you must also include your Adafruit IO username and key. Copy the following example, paste it into your settings.toml file, and update:

- `your_wifi_ssid`
- `your_wifi_password`
- `your_aio_username`
- `your_aio_key`

```
CIRCUITPY_WIFI_SSID = "your_wifi_ssid"
CIRCUITPY_WIFI_PASSWORD = "your_wifi_password"
ADAFRUIT_AIO_USERNAME = "your_aio_username"
ADAFRUIT_AIO_KEY = "your_aio_key"
```

Some projects use different variable names for the entries in the **settings.toml** file. For example, a project might use `ADAFRUIT_AIO_ID` in the place of `ADAFRUIT_AIO_USERNAME`. **If you run into connectivity issues, one of the first things to check is that the names in the settings.toml file match the names in the code.**

> Not every project uses the same variable name for each entry in the settings.toml file! Always verify it matches the code.

settings.toml File Tips

Here is an example **settings.toml** file.

```
# Comments are supported
CIRCUITPY_WIFI_SSID = "guest wifi"
CIRCUITPY_WIFI_PASSWORD = "guessable"
CIRCUITPY_WEB_API_PORT = 80
CIRCUITPY_WEB_API_PASSWORD = "passw0rd"
test_variable = "this is a test"
thumbs_up = "\U0001f44d"
```

In a **settings.toml** file, it's important to keep these factors in mind:

- Strings are wrapped in double quotes; ex: `"your-string-here"`

- Integers are **not** quoted and may be written in decimal with optional sign (`+1`, `-1`, `1000`) or hexadecimal (`0xabcd`).
    - Floats, octal (`0o567`) and binary (`0b11011`) are not supported.
- Use `\u` escapes for weird characters, `\x` and `\ooo` escapes are not available in **.toml** files
    - Example: `\U0001f44d` for (thumbs up emoji) and `\u20ac` for € (EUR sign)
- Unicode emoji, and non-ASCII characters, stand for themselves as long as you're careful to save in "UTF-8 without BOM" format

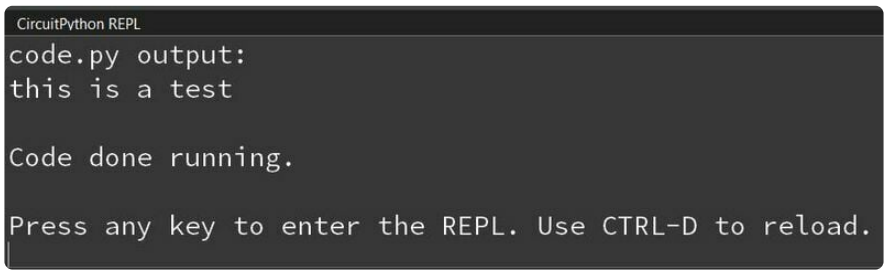
When your **settings.toml** file is ready, you can save it in your text editor with the **.toml** extension.

Accessing Your **settings.toml** Information in **code.py**

In your **code.py** file, you'll need to `import` the `os` library to access the **settings.toml** file. Your settings are accessed with the `os.getenv()` function. You'll pass your settings entry to the function to import it into the **code.py** file.

```
import os

print(os.getenv("test_variable"))
```



In the upcoming CircuitPython WiFi examples, you'll see how the **settings.toml** file is used for connecting to your SSID and accessing your API keys.